U N I V E R S I T Y O F T A R T U

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Institute of Computer Science

Speciality of Computer Science

Dmitri Melnikov

F2F Computing as a Base for Network Games

Bachelor's thesis (4 cp)

Supervisor: Ulrich Norbisrath, PhD

Author: ..... “.....“ June 2009

Supervisor: ..... “.....“ June 2009

Allow to Defense:

Professor .... “.....“ June 2009

TARTU 2009

Contents

Introduction

Computer technologies are a part of our everyday life. A Web browser, an email client, an instant messaging software – these are all instruments of an average Internet user. Such typical users have a certain computational power at their disposal, a certain variety of interests and a certain circle of friends with whom they interact on a daily basis. It is possible to share and exploit the available resources by integrating popular instant messaging software with a special framework. Using this framework users can submit their jobs to their friends in a manner similar to submitting a job to a *Grid*.

In this thesis I shall describe the process of porting a standard client-server software to the *Friend-to-Friend* (F2F) [1, 8] computing platform using the game *The Bub's Brothers* [2] as an example. A detailed explanation of both F2F and client-server architectures is presented. The resulting application should provide the same features as the original and it should not be apparent to the average user whether he is using the modified version or the original.

Grid computing [7] is a form of distributed computing where different resources (possibly scattered around the world) such as CPU units, memory, storage space, special devices can be combined in one system to perform a given task. Grid contrasts with conventional supercomputers that are limited to their own hardware however great it might be. As a Grid is composed of many different resources its size is not fixed, it is designed to grow and shrink and be as flexible as possible. While in the beginning Grid computing was used primarily by scientists for their computationally intensive tasks, the situation has been changing rapidly in the past few years.

If we now consider the Internet users with their resources as a model for a Grid, we can try to imagine a system where everyone profits. Another important concept is *Peer-to-peer networking*. In a peer-to-peer network everyone is equal, there are no clients or servers because a peer is both a client and a server and there is limited or no centralized control. The great thing about such a network is that it is always changing, peers come and go and the network appears to have a life of its own. Peer-to-peer networks can be used for various different tasks such as file sharing which is their most known use today.

Combining these two models in one a new concept is created: a network that allows sharing resources between equal peers. If a connection to such a network could be easily made for all those average users then many would benefit from such a wealth of resources. Fortunately a software to accomplish this already exists – *Friend-to-Friend computing* or *F2F*. F2F uses instant messaging (IM) to deliver messages between the members of the group, which is logical since nearly everyone has some sort of IM client installed on their local computer. The group consists of people from the person's IM contact list and we shall call members of such a list *friends*.

It is clear that the variety of software that can be made using F2F is almost limitless. This comes from the fact that users have different interests – while some would like their friends to help them with their scientific calculations, others just want to play a game (or maybe do both at the same time). One of the problems we encounter is that there is a lot of software which is not ready for F2F computing. The thesis aims to give an overview of how porting applications to F2F can be conducted.

Since the Bub's Brothers game (a modified version called *Mullivelled*) has already been used in an Educational Gaming project [9] caried out at the University of Tartu the secondary goal of the thesis is to give judgment on how F2F can be used to promote cooperation and collaboration between users.

1. The Bub's Brothers

1.1 The Game

Bub's Brothers [2] is an open source multi player arcade game written in the python programming language by Armin Rigo. The game is a direct clone of the famous and highly successful *Bubble Bobble* game first released in 1986 and later ported to numerous platforms and followed by a series of sequels. The players control small animated dragons of different colors and pass from one level to the other collecting items and throwing bubbles at their enemies. The goal is to complete all levels and earn as much points as possible.

Figure 1.1 The Bubs Brothers with a single player

The game features many items that can change the behavior of the dragon, for example allow it to run faster or jump higher. Another interesting feature are the mini-games (i.e. Tetris, pacman) triggered for a short time by a special item. Such unexpected events contribute to the positive gaming experience.

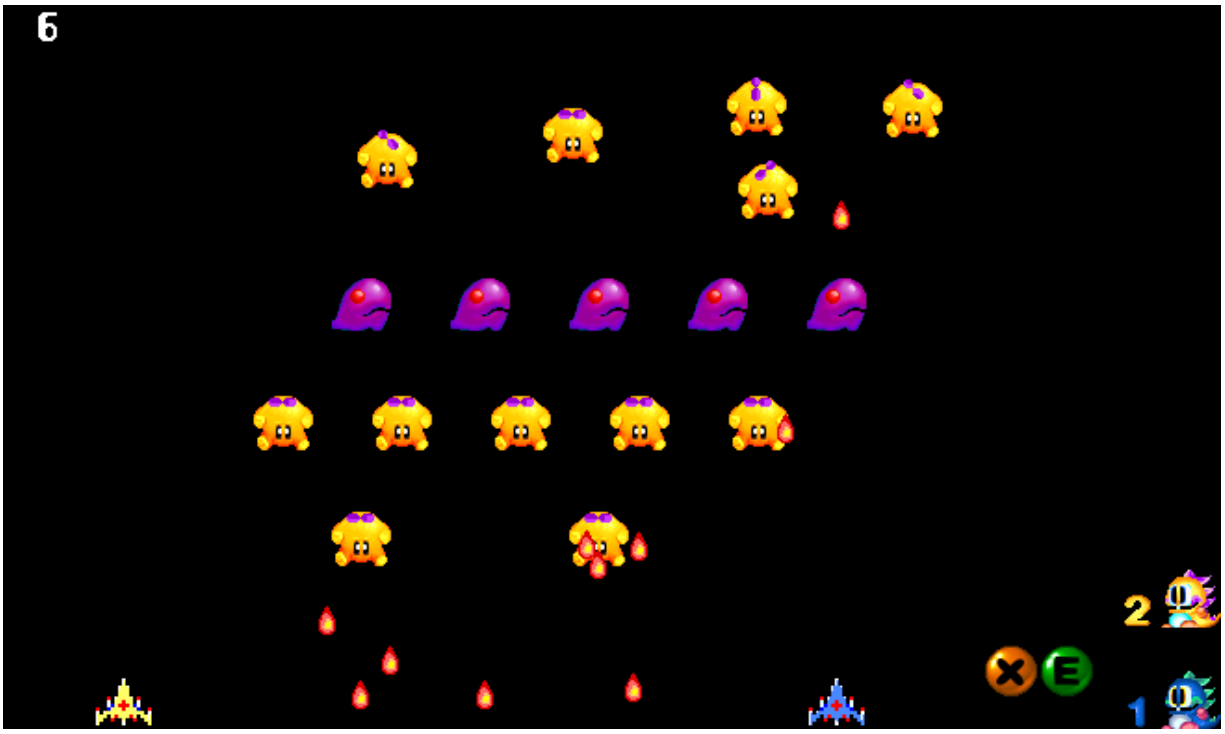


Figure 1.2 Galaga mini-game with two players

The game's strong side is it is support for multiple players. Up to 10 players can enjoy the game together making it even more fun to play.

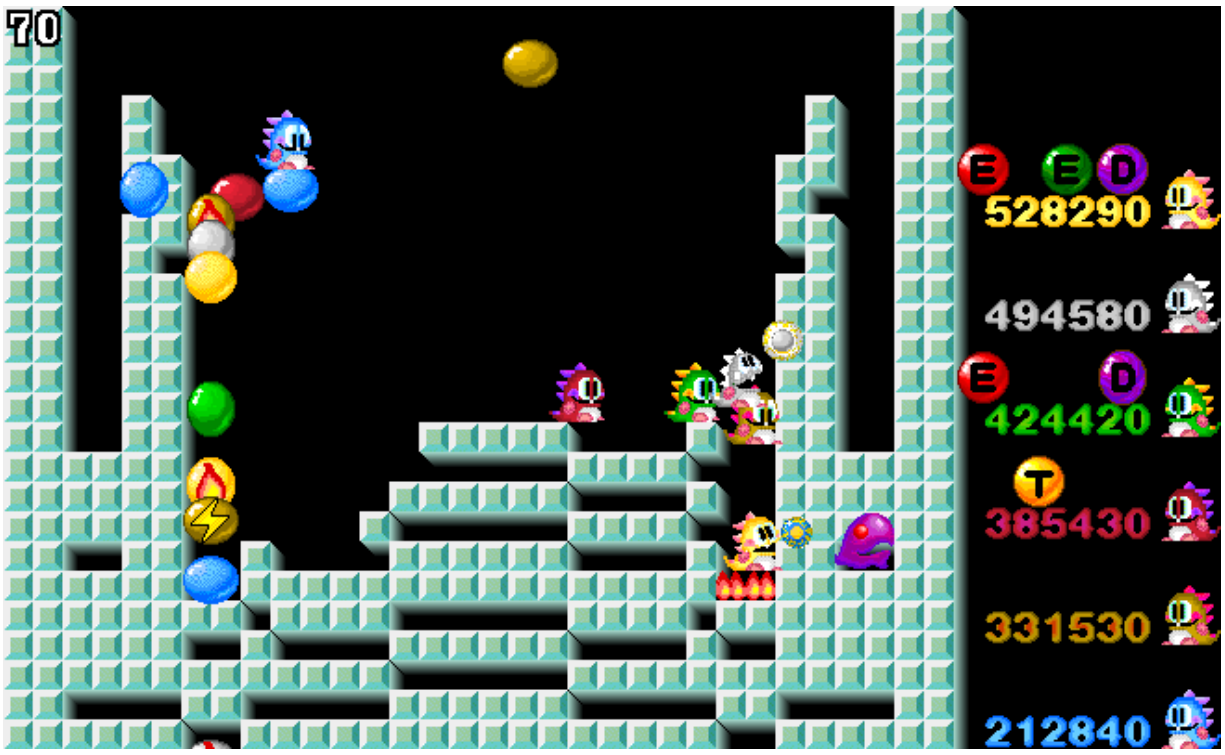


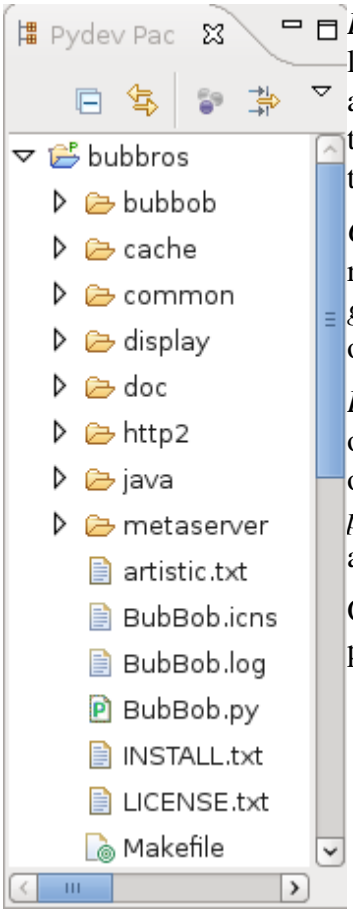
Figure 1.3 A game with 6 players

1.2 The game architecture

To understand how the game is made it is best to start with a quick overview of different packages and files that are in use.

BubBob.py – a wrapper program that starts the game server, the meta-server and launches the users browser.

In this game the meta-server is used to set up and control the main server with a browser. For example a user can change different levels, start a new game, join an existing game and do other configuration related work. It is also possible to register the server on the Internet and allow anyone to join. I shall be using the gaming server directly, so the meta-server shall not be discussed further.



Bubbob/ - contains all the necessary data for the game such as levels and images. Many python scripts that compose the game are also located here: *boards.py, bonnuses.py, images.py,* etc. Because the networking code is not located in these scripts, the changes to these files in the modified version are minimal.

Common/ - scripts with networking and binary data manipulation reside here. Also the protocol messages are defined in *msgstruct.py*. *gamesrv.py* is of particular interest as it contains most of the logic of the game server and shall be modified to a great extend.

Display/ - scripts that actually render the game to the screen using one of the possible graphical output systems such as *PyGame, GTK* or *Java*. And also the networking code for the client, mostly in *pclient.py,* which also has to be modified when moving the application to the F2F platform.

Other subdirectories do not contain important data for the current project and shall not be discussed here.

Figure 1.4 The game structure

2. Client-server architecture

This chapter gives an overview of the architecture, the algorithms and presents the initial message exchange between the client and the server.

2.1 Overview

In a a typical client-server architecture the client sends requests for services to the server and the server responds by providing the requested services. Here, a service can be anything from a simple message to a large object of data. Clients and servers usually run on separate computers but can also reside on the same machine (*single seat* setup) and use a specially defined language or *protocol* to communicate. A web browser and a web server are a good example of such an architecture.

To facilitate the process of moving the application to the F2F platform we first describe how the client and the server interact in the original game.

2.2 Bub's Brothers server

The main classes that are used by the server are called *Game* and *Client*, both contained in *common/gamesrv.py*.

The Game class is responsible for opening the data socket, the ping socket, the http socket and the broadcast socket but also processes new incoming connections, calculates the update intervals and send the game's update data to all the clients.

The Client class represents a connection to a specific client and has methods for sending and receiving data and also different kinds of handling function that correspond to the defined protocol messages.

The heart of the server is the *mainloop()* function, its algorithm is the following:

1. Looping condition: While there are elements in the list of sockets. The sockets are created in the Game class and every time a client connects a new socket is added to the list.
2. A delay interval is computed using the Game class. A delay is based on the time required by the server to generate a frame.
3. A *select* function is called with all the sockets and the calculated delay as arguments. Select is part of the standard networking API provided by python, its job is to return all the sockets that are ready for input, output or are in the state of error. If the delay argument is provided then the call to select blocks for no more that the value of the delay and returns empty lists if nothing is ready.

4. For all the sockets ready with input, a handler assigned to that socket is called. For the client socket this handler would receive and process the data.
5. Handle errors and exceptions. If something unexpected has happened the server disconnects the problematic client, or may even shutdown itself.

Algorithm 2.1 The server's main loop

2.3 Bub's Brothers client

The client is located in *display/Client.py* and *display/pclient.py* and is represented by the *Playfield* class.

The Playfield class is responsible for sending, receiving and updating the data on the screen. It contains handler functions that know how to respond to different protocol messages. It also takes input from the user's keyboard and mouse.

The central function of the client is (again) called *mainloop()* and its algorithms is as follows:

1. Loop condition: infinite loop
2. Process the keys pressed by the user.
3. Call select on the input socket with a default delay.
4. Process the keys pressed one more time.
5. If the input socket is ready, receive the message, decode and process it using the assigned handler.
6. Check udp sockets that are also maintained by the client. Broadcast messages and data can be sent and received via udp sockets.
7. If any data was transferred along with the protocol message, then use it to update the game.
8. Call the graphical system's update method to actually render the updated game state to the screen.
9. Possibly update the default delay with the value returned by the previous rendering call.

Algorithm 2.2 The client's main loop

2.4 Bub's Brothers client-server interaction

The interaction begins when the select call in the server's *mainloop()* indicates that the listening tcp socket is ready. This means that a new client is trying to make a connection. Having checked that the existing client number does not exceed the maximum, the server proceeds to accept the incoming connection by creating an instance of the Client class. After this the following messages are exchanged:

From	To	Message	Handled in
Server	Client	MSG_WELCOME	Playfield.__init__
Server	Client	MSG_DEF_PLAYFIELD	Playfield.msg_def_playfield
Server	Client*	MSG_PLAYER_JOIN	Playfield.msg_player_join
Client	Server	CMSG_PROTO_VERSION	Client.protocol_version
Client	Server	CMSG_PING	Client.ping
Server	Client	MSG_PONG	Playfield.pong

- * - the message is sent to all the active clients

Table 2.1 The initial message exchange between the client and the server

The protocol message names are self-describing with messages sent from the client prefixed with the letter 'C'. As observed from the table 2.1 the client messages are handler inside the *Playfield* class and the server messages inside the *Client* class. During the initial message exchange the MSG_PLAYER_JOIN message is sent to all active participants so that everyone is notified about the new player. When these messages are exchanged and processed the new client is initialized and ready to receive messages with binary data and the game can begin.

3. Friend-to-Friend architecture

3.1 Overview

The main idea of Friend-to-Friend computing is to use Instant Messaging to set up a Grid between people in the same contact list [6]. Most IM systems allow users to create a chat group by inviting selected friends. F2F uses such group resources to build a computational Grid (sometimes abbreviated as *frid*).

3.2 Instant Messaging

Instant Messaging (IM) is important for F2F because the IM layer delivers the actual messages and data.

IM technologies make real-time communication possible between two or more participants. The idea of IM is much closer to that of *Internet Relay Chat* (IRC) than to e-mail because messages are sent instantly with a minimum delay. However with IM the information about the contacts is stored in the *contact list* and there is no need to join a channel just to see who is there. The procedure of logging on to the IM server is not difficult and requires no special knowledge, with most clients only a user name (or e-mail address) and a password is required. One of the advantages of IM are the users' statuses. For example, if a user's status is set to “Away” then he probably is not present at the moment.

There are many different IM networks that define their own protocols, for example MSN Messaging, Jabber, ICQ. The IM clients are also different, some provide access to only one IM network (MSN Messenger, ICQ client) while others can communicate with many (Pidgin, SIP Communicator). Some of these clients also have F2F support.

3.3 F2F framework

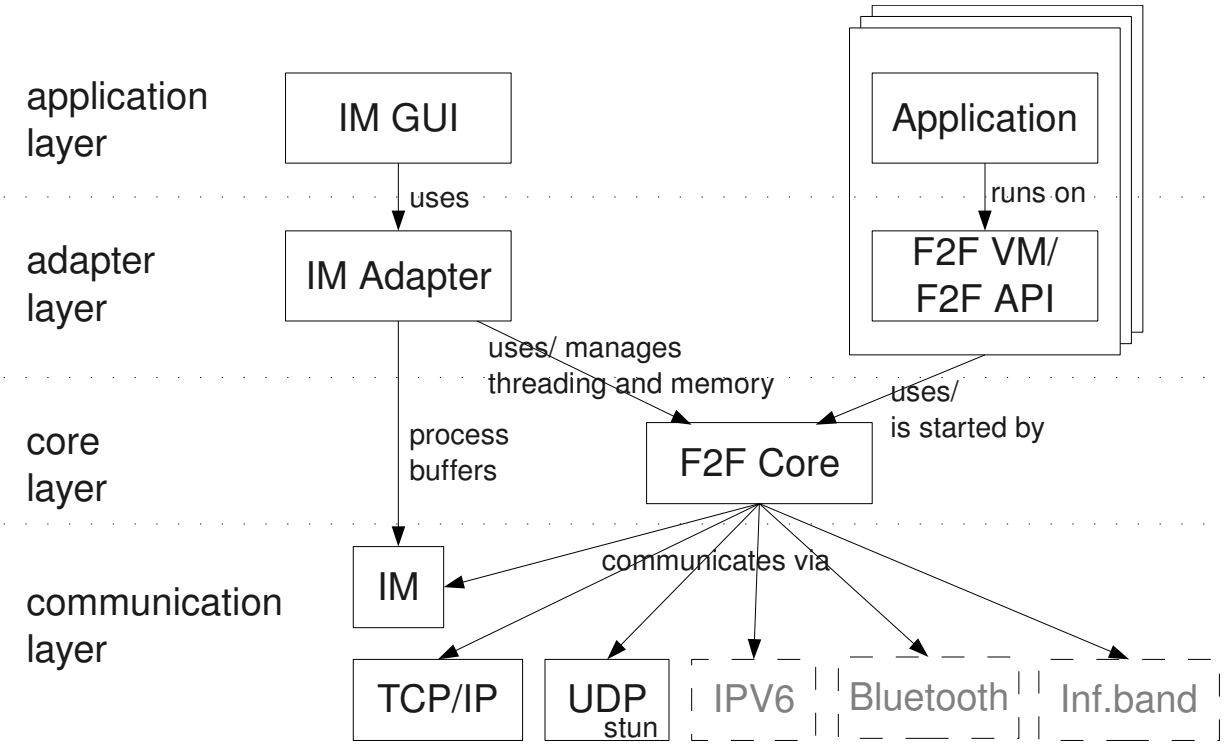


Figure 3.1 F2F architecture

The framework can be divided into four layers: application, adapter, core and communication layer. The task that requires computation is considered to be in the application layer. The task can use the F2F API or run on the F2F virtual machine, which is in the adapter layer, used for managing threading and memory. The core layer processes messages forwarded by the adapter. The communication layer is used to exchange messages. The design of the communication layer allows it to use different channels for transferring messages. One of these channels is obviously IM but direct TCP and UDP can be also considered. The goal is to make this layer as robust and dynamic as possible so that the best connection can be used.

3.4 F2F API for Python

Because the Bub's Brothers game itself is written in python it is convenient to use the provided python F2F application programming interface (API) when making changes to the game. The python API is part of the *F2FAdapter* project called *F2FHeadless* and it depends on another project called *F2FCore* containing the f2f core functionality. F2FCore is written using the C programming language and the python bindings are created by the *SWIG* [4] application automatically during the build process.

The API is small and has only two classes and two global functions, all contained in the **f2f** package:

Class *Peer:*

- ***getUid****()* - returns the identification of t he peer represented by the class instance.
- ***equals****(otherpeer)* - returns true if *otherpeer* and the class instance are the same, false otherwise.
- ***send****(group, obj)* – send data referenced by *obj* to the peer represented by the class instance, a group must also be specified.
- Other methods of the class are used internally and should not be called directly.

Class *Group:*

- ***getUid****()* - returns the identification of the group represented by the class instance.
- ***equals****(othergroup)* - returns true if *othergroup* and the class instance are the same, false otherwise.
- ***getPeers****()* - returns a list of all the peers belonging to this group
- Other methods of the class are used internally and should not be called directly.

f2f.myPeer*()* - returns an object of the Peer class representing the caller.

f2f.receive*([timeout])* - blocks until data is available on the internal stack, then returns the data. If the *timeout* argument is provided, waits no longer than the value of *timeout*.

4. Moving to F2F

4.1 General approach

The goal is to move the Bub's Brothers game to the F2F computing platform by substituting the standard networking code with F2F API. In a F2F network all friends are considered to be equal and any of them could be a client, a server or even both. However, in the current application it would be opportune for one friend to act as a server and all others as clients. Using this model minimum changes have to be made in the existing application to transfer it to the F2F platform.

The starting point is to understand how the application works from the moment the user starts the game. The game has a lot of functionality unnecessary for F2F computing such as network configuration and meta-server and it is important to identify these components so that we could later remove them.

The game server uses four different sockets to send and receive data: the data socket, the ping socket, the http socket and the broadcast socket. The code responsible for opening the sockets is defined in *openXsocket()* functions, where X is one of {*http*, *ping*, *tcp*, *broadcast*}. However, opening a socket is not enough by itself. In a large application we also need a easy and convenient way to manage the sockets, to add, find and remove them. The original Bub's Brothers game keeps the following structures:

- `socketsbyrole[role] = socket` – the socket is referenced by a role, a role can be one of *LISTEN, PING, HTTP, BROADCAST* or *CLIENT*.
- `serversockets[socket] = handler` – this data structure sets the socket handler (a function that reads from the socket when it is ready)
- `socketports[socket] = port` – here the socket's port number is stored

When the application requires for example a socket for listening to new incoming connections, it can be located with **`findsocket`**`('LISTEN')`. Similarly, sockets can be added and removed by their role.
Because F2F does not use sockets directly and provides no mechanisms to handle the sockets, all of the socket management and creation code has to be removed.

Typical input and output operations with sockets are substituted by the F2F API:

- For receiveing `socket.`**`recv`**`()` was originally used. `f2f.`**`receive`**`()` can be used as a replacement.
- For sending `socket.`**`send`**`(data)` or `s.`**`sendall`**`(data)` was used. The difference between the two functions is that `sendall` continues to send data until everything is sent. Both can be replaced with `f2fpeer.`**`send`**`(f2fgrp, data)`.

4.2 Challenges

Not everything is directly replaceable and difficulties still occur. The problems are caused both by the simplicity of the F2F API and the differences in approaches of F2F and standard sockets. There is no generic algorithm for moving an application to the F2F platform, every program has to be treated separately. Here we describe some of the common problems that came up in the process of moving the Bub's Brothers game to F2F along with their possible solutions. In similar applications the problems can be the same.

- Connection to the server.

  In the original game, as seen in *Table 2.1*, the first protocol message comes from the server to the client. This is because the server knows about the new client when it makes a socket connection. Having made the connection, the server then sends the first protocol message to the client. Meanwhile the client is blocked and cannot do anything until it receives this message.

  With F2F such an approach is not possible because the server does not know of the client unless the client sends something. The server just waits for data to come and only when the receive call returns something can the server recognize the sender.

  A possible solution is to send the first message from the client itself to the server. For the game's protocol a new message was introduced – `MSG_CONNECT`. After receiving this message the server remembers the new client and the message exchange sequence described in *Table 2.1* begins.

- Select call.
  In the original game code a select function call is used to determine which sockets are ready for reading data. If the delay argument is provided then select will block not longer than the value of the delay and return an empty list if no sockets are ready. F2F API originally had no way to imitate the select call with a timeout.
  However, it can be simulated by modifying the existing `f2f.receive()` function. `f2f.receive()` works by checking in a loop whether a local message stack is empty. When a message is available it returns this message. By adding a delay argument we can create a function that returns by the end of the timeout even if nothing was received. Such function can effectively substitute the select call.

- Paths.
  In the original game, when the game was started the executing code would change the current working directory (with a `os.chdir` call). This was necessary because the game files were referenced with relative paths. This approach does not work however with the current F2F setup. Because the game is a F2F job, it is not run by the user but from the F2F client code. Since the calling code has its own dependencies changing the current directory in the game would also change the paths to these dependencies.

A possible solution that was used in this project is to change all the relative paths to absolute ones.

- Debugging.
  Finding problems in a network application is always difficult especially when binary data is transferred. With F2F it can be even more challenging because it is not obvious for the programmer how and when the data is actually delivered. Here it is important to keep the message exchange logic the same as it was before and to be careful when introducing new protocol messages and their handlers. Current F2F implementation has a buffer of limited size for sending, however this did not turn out to be a problem for the current project.

4.3 Resulting application

As the python version of F2F is not yet integrated with an IM client, we shall be using the F2FHeadless project to represent the peers in the IM group. The peer who starts the group and submits the job to everyone else is called the *initiator* or *master*. The initiator's friends are called *slaves.* There can be many slaves but only one master. Finally we must wrap our game in a special job script that can be used for sending.

The resulting application requires several new python scripts to run: *f2fMaster.py, f2fSlave.py, bubs.py.*

- f2fMaster.py represents the friend that initiates the job (the initiator). The initiator has a list of friends (slaves) that he can form a computation group from. When any of those friends become connected to the IM server, the initiator automatically sends them the job. In our application the initiator shall be acting as the game server, however this is only a matter of choice and the initiator can also be the game client. The script is parametrized with the list of slaves and can be used from the command line.
- f2fSlave.py represents the friend of the initiator that receives the job and starts executing it. In our application all initiator's friends shall be game clients. There can be many scripts of this kind, each representing a slave, for example we can create f2fSlave2 and f2fSlave3 scripts and their contents will be almost the same. The slave scripts are parametrized with the list of masters and can be run from the command line.
- bubs.py is a wrapper around the our game application. It decides what code must the master and slaves execute. Currently for the slaves a game client is started, while in case of the master a game server is run.

The implementation of all of these new scripts is simple, f2fMaster and f2fSlave only need to create a F2FHeadless object by specifying their IM user name and password, the list of friends (or in case of slaves only the initiator) and the job to execute – bubs.py.

The game was modified using the *Eclipse IDE's Pydev* plug-in. Eclipse provides a powerful development environment that facilitates navigating, managing and editing source code. Also

the run targets can be added. In the development setup the run targets were set to invoke the *scons* building tool that would compile the F2FCore project and run *swig* to create python bindings from C code. When scons finishes Eclipse runs the f2fMaster or f2fSlave scripts.

The IM server that was used for testing is *Openfire 3.6.3* [5]. Openfire is an open source Jabber/XMPP server written in Java that has a Web-based administration panel for managing users and server settings. Thus whenever the f2fMaster or f2fSlave scripts are executed and the F2FHeadless object is created, a user logs onto the Openfire IM server via the Jabber protocol.

Ideally the game could now be started from an IM client by sending it to all the peers willing to play. However, the F2F for python is not yet integrated with any IM client and integration is not a part of this project.

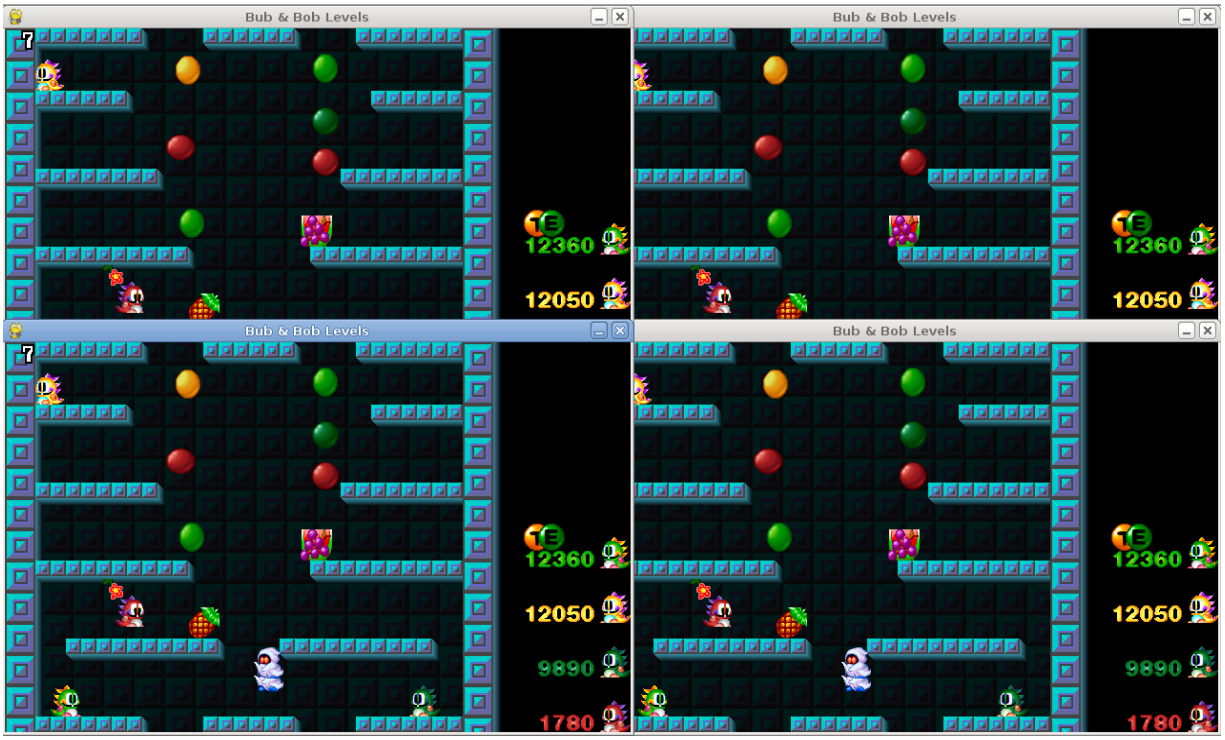


Figure 4.1 Four clients launched from one machine connected over F2F

5. F2F in Educational Gaming

5.1 Educational Gaming

One of the most valuable skill of a good specialist is the ability to communicate. Surprisingly this fact if often overlooked and many computer science students and other professionals working full time with computers fail to express and share their thoughts with others. Educational gaming [9] is aimed at promoting cooperation and collaboration between students by providing an suitable environment for communication. The idea of the project is to use computer games for learning. Dividing the students into groups and assigning each group a certain goal requiring cooperation, active participation can be expected.

From the technical side this project brings up certain criteria that an environment has to meet.

Firstly a game has to be both fun and simple and also provide multi player capabilities. Secondly there has to be enough room for cooperation in the game between players.

Bub's Brothers seems to be a suitable candidate for such a project. A modified version of the game with specially designed levels has been created and called *Mullivelled* [3].

The gaming itself was successful however the technical synchronization and administration proved to create a challenge. The gaming setup requires a new approach that would allow fast and effortless environment creation.

5.2 Integration with F2F

The F2F framework can potentially solve the occurring problems. With F2F integrated into a IM client for the playing to begin all it takes is to create a IM group and submit a game. With such a setup users do not even have to have a copy of the game. It is important that users focus only on the gaming and not pay attention to the technical side.

The idea to use F2F in educational gaming is purely theoretical at the current moment and no experiments have yet been conducted. Before any testing can take place it is necessary for the F2F to be integrated with the chosen IM client.

F2F computing has the potential to facilitate the setup of the gaming environment however much work is still to be done in this direction.

6. Summary

Modern computers have much resources at their disposal. Average users however do not use most of their resources in their daily computer activities. Using a framework integrated with popular software such as an IM (*Instant Messaging*) client we can put some of that computational power to work. A form of a *grid* can be constructed from the people in the message list. Such a grid is different from the standard heavyweight high-end architecture usually associated with the word grid. It possesses features of a Peer-to-Peer network and is more dynamic. The framework that is aimed at achieving this goal is called Friend-to-Friend (*F2F*) computing.

There is a lot of good software already written for all kinds of purposes. To use it on the F2F platform it is necessary to either run the application inside the virtual machine or to use the F2F Application Programming Interface (*API*). In the first case the program requires no modification whereas in the second manual changes are necessary. For this thesis the game called Bub's Brothers shall be modified to use the F2F API.

Bub's Brothers is a multi player arcade game written in Python. It is a remake of the popular Bubble Bobble game released in the 80s. The game is a typical client-server architecture that uses standard sockets to exchange data. A step-by-step description of the server and client algorithms is provided. The game is well-written and uses strict folder structure to separate networking, game logic and display.

I began moving the game to the F2F platform by identifying both the code that requires substitution and the code that becomes unnecessary in the new application. The original game server and client both have more than one socket opened for various purposes. All sockets were removed and send/receive calls substituted with the corresponding F2F API equivalents. Thus all messages are exchanged using only a single F2F channel. Some of the challenges that came up during the modification process are considered along with their possible solutions. Finally, I present a F2F-ready game with a description of the environment it was tested in.

Another modified version of the same Bub's Brothers game called *Mullivelled* has been used for the Educational Gaming project at the University of Tartu. The project requires a platform that would facilitate the game administration and synchronization between players. The adapted version of Bub's Brothers will greatly simplify the adminstration tasks of Mullivelled and achieve better game performance and experience.

Appendix A Source code

The source code of the project can be found at the following address:
http://ulno.net/f2f/interactive/bbros

The archive contains both F2FCore and F2FHeadless projects that can be easily imported into Eclipse. The modified game itself is located in F2FHeadless/src/tests/bbgame. After importing the projects into Eclipse, please follow the instructions at http://ulno.net/f2f/development to configure the building process and run targets.

F2F baseeruvad võrgumängud

Bakalaureusetöö (4 ap)

Dmitri Melnikov

Resümee

Tänapäeval tavakasutaja arvutil on olemas palju ressursse mida ta ei kasuta oma igapäevases töös. Tavaliselt aga peaaegu igaühel on juba paigaldatud enamlevinud programmid nagu veebibrauser ja kiirsuhtluse klient. F2F (*Friend-to-Friend*) raamistik annab ligipääsu sellistele ressurssidele kasutades populaarseid suhtlusprogramme. Võrreldes *Grid* süsteemiga on F2F dünaamilisem ja kergekaalulisem.
Tarkvara käivitamiseks F2F platvormil on kaks võimalust: virtuaalmasina sees või kasutades F2F API. Virtuaalmasina korral ei pea tarkvara muutma, F2F API nõuab aga olemasoleva koodi asendamist. Kuna F2F virtuaalmasin ei ole veel implementeeritud, keskendume teisele variandile. Töö eesmärgiks on näidata kuidas saab rakenduse F2F platvormile üle viia *Bub's Brothers* mängu näitel.
Bub's Brothers on multi-mängija arcade kus üheaegselt saavad mängida kuni 10 mängijat. Rakendus on kirjutatud Python keeles ja kujutab endast tüüpilist client-server arhitektuuri. Töö annab ülevaade nii olemasolevast võrgukihist kui ka uuest F2F kihist. Üleviimise protsessil tekkinud probleemid on lahti seletatud koos nende võimalike lahendustega. Lõpus on toodud valmis F2F rakenduse ja testimise keskkonna kirjeldus.
Bub's Brothers modifitseeritud versioon *Mullivelled* oli juba kasutatud ühe projekti raames mille eesmärk oli arendada tudengite koostööd. Projekt nõuab platvormi millega saab kergesti mänge administreerida ja sünkroniseerida. Sellise platvormi realiseerimiseks on mõistlik kasutada F2F võimalusi.

References

[1] Friend-to-Friend (F2F) Computing. http://ulno.net/f2f

[2] The Bub's Brothers. http://bub-n-bros.sourceforge.net

[3] Mullivelled. http://ulno.net/algs/GameOrchestration

[4] Simplified Wrapper and Interface Generator, SWIG. http://www.swig.org

[5] Openfire Server. http://www.igniterealtime.org/projects/openfire/index.jsp

[6] Keio Kraaner. *A Framework for Friend-to-Friend Computing*. June, 2008

[7] F. Berman, G.C. Fox, and A.J.H. Hey. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[8] U. Norbisrath, K. Kraaner, E. Vainikko, and O. Batrasev. *Friend-to-Friend Computing - Instant Messaging Based Spontaneous Desktop* Grid. In The Third International Conference on Internet and Web Applications and Services (ICIW 2008), 2008.

[9] Ulrich Norbisrath, Ivar Männamaa, Anne Villems, Külli Kalamees-Pani : *Mullivelled - Wrapping Computer Games into Educational Gaming Environments.* Submitted and accepted for International Gaming and Simulation Association Conference (ISAGA 2008), July 2008, Kaunas/Lithuania